Programming Languages and Machine Learning

Martin Vechev
DeepCode.ai and ETH Zurich

DEEPCODE

ETH Zürich

PL Research: Last 10 years (sample)

- (Semi-) Automated Program Synthesis
  - Mostly learning functions/algorithms over discrete spaces (from examples, natural language, components, partial specs, etc)
- Automated Symbolic Reasoning
  - Abstract Interpretation = theory of sound & precise approximation
  - SMT solvers
- Approximate/Probabilistic Programming
  - Applications/Analysis/Synthesis

Two part talk (22 + 3) ☺

Learning-based Programming Engines

SLANG | JS NICE | NICE 2 Predict | DEGUARD | Deep3

1. Pick a structure of interest, e.g., trees:

2. Define a DSL for expressing functions:
(can be Turing complete)

```
TCond   ::= ε | WriteOp TCond | MoveOp TCond
MoveOp  ::= Up, Left, Right, DownFirst, DownLast,
            NextDFS, PrevDFS, NextLeaf, PrevLeaf,
            PrevNodeType, PrevNodeValue,
WriteOp ::= WriteValue, WriteType, WritePos
```

3. Synthesize $f_{best} \in$ DSL from Dataset D:

$$f_{best} = \underset{f \in \text{DSL}}{\text{argmin}}\ \text{cost}(\boldsymbol{D}, f)$$

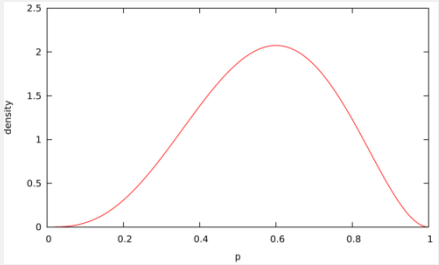
4. Use f_{best} on new structures:

f_{best} () → γ

http://plml.ethz.ch

PSI: Exact Solver for Probabilistic Programs

```
def main() {
 p := Uniform(0,1);
 r := [1,1,0,1,0];
  for  i in [0..r.len] {
  observe(
     Bernoulli(p) == r[i]));
 }
 return p;
}
```



http://psisolver.org

Two part talk (22 + 3) ☺

Learning-based Programming Engines

SLANG | JS NICE | NICE 2 Predict | DEGUARD | Deep3

1. Pick a structure of interest, e.g., trees:

2. Define a DSL for expressing functions:
(can be Turing complete)

```
TCond   ::= ε | WriteOp TCond | MoveOp TCond
MoveOp  ::= Up, Left, Right, DownFirst, DownLast,
            NextDFS, PrevDFS, NextLeaf, PrevLeaf,
            PrevNodeType, PrevNodeValue,
WriteOp ::= WriteValue, WriteType, WritePos
```

3. Synthesize $f_{best} \in$ DSL from Dataset D:

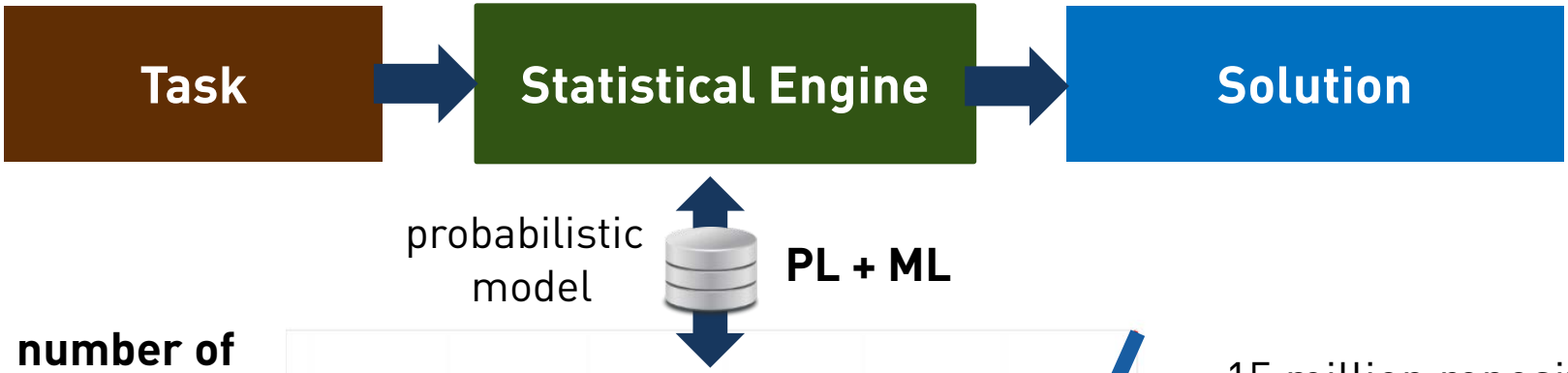
$$f_{best} = \underset{f \in \text{DSL}}{\text{argmin}}\ \text{cost}(\boldsymbol{D}, f)$$

4. Use f_{best} on new structures:

f_{best} () → γ

http://plml.ethz.ch

Probabilistic Learning from Code



15 million repositories

Billions of lines of code

High quality, tested, maintained programs

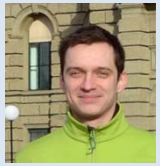
Probabilistic Learning from Code

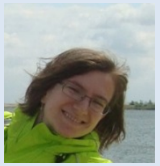
Probabilistically likely solutions to problems hard to solve otherwise

Joint work with :


Veselin Raychev


Andreas Krause


Pavol Bielik

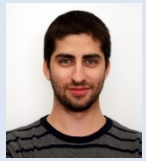

Christine Zeller


Svetoslav Karaivanov


Pascal Roos


Benjamin Bischel


Timon Gehr


Petar Tsankov


Mateo Panzacchi

Publications

- Program Synthesis for Char. Level Language Modeling, **ICLR'17 sub**
- Learning a Static Analyzer from Data, **https://arxiv.org/abs/1611.01752**
- Statistical Deobfuscation of Android Applications, **ACM CCS'16**
- Probabilistic Mode for Code with Decision Trees, **ACM OOPSLA'16**
- PHOG: Probabilistic Mode for Code, **ACM ICML'16**
- Learning Programs from Noisy Data, **ACM POPL'16**
- Predicting Program Properties from "Big Code", **ACM POPL'15**
- Code Completion with Statistical Language Models, **ACM PLDI'14**
- Machine Translation for Programming Languages, **ACM Onward'14**

Statistical Engines

apk-deguard.com

DEGUARD

DEEP3

jsnice.org

JS NICE

nice2predict.org

SLANG

NICE 2 Predict

more: http://plml.ethz.ch

JSNice.org

JS NICE

✓ Every country ✓ ~200,000 users ✓ Top ranked tool

This Page Amsterdam @thispage_ams · Jul 16
Do you write ugly JavaScript code? Not to worry. JSNice will make it look like you are a superstar coder. Yai! - buff.ly/1HR4JL7

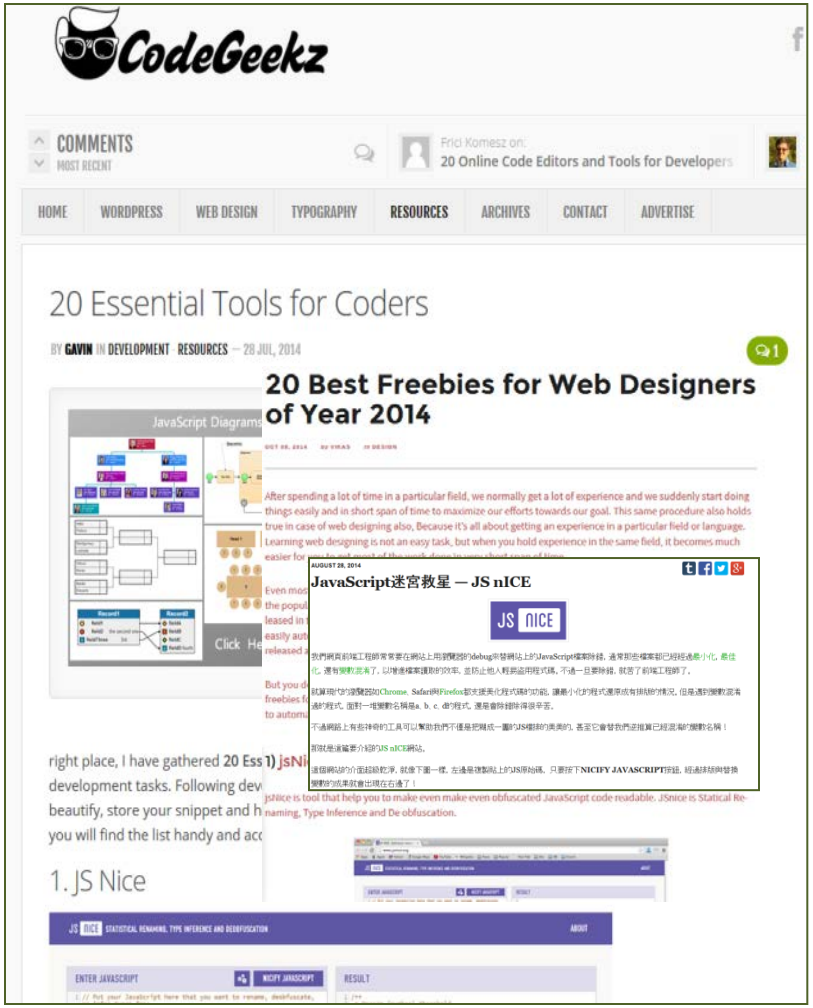
Ingvar Stepanyan @RReverser · Aug 6
JSNice.org became my must-have tool for code deobfuscation.
Expand Reply Retweet Favorite More

Brevity @seekbrevity · Jul 28
JSNice is an a amazing tool for de-minifying #javascript files. JSNice.org, Its great for #learning and reverse engineering.
Expand Reply Retweet Favorite More

Alvaro Sanchez @alvasavi · Jun 10
This is gold.
Statistical renaming, Type inference and Deobfuscation.
jsnice.org
Expand Reply Retweet Favorite More

Alex Vanston @mrvdot · Jun 7
I've been looking for this for years: JS NICE buff.ly/1pQ5qfr #javascript #unminify #deobfuscate #makeItReadable
Expand Reply Retweet Favorite More

Kamil Tomšík @cztomsik · Jun 6
tell me how this works!
de-minify #jquery #javascript incl. args, vars & #jsdoc
impressive! jsnice.org

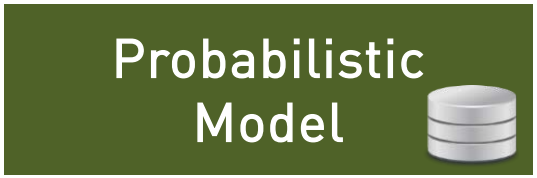


A Key Question

Data

Learning

Model



Widely Applicable	Efficient Learning	High Precision	Explainable Predictions

Training dataset ***D***

```
f.open(“f2” | “r”);
f.read();
```

```
f.open(“f2” | “w”);
f.write(“c”);
```

```
f.open(“f1” | “r”);
f.read();
```

query:

```
f.open(“file” | “r”);

f. ?
```

Training dataset ***D***

```
f.open(“f2” | “r”);
f.read();
```

```
f.open(“f2” | “w”);
f.write(“c”);
```

```
f.open(“f1” | “r”);
f.read();
```

query:

```
f.open(“file” | “r”);

f. ?
```

3-gram model on tokens

Hindle et. al., ACM ICSE’12

```
P(open  | f. ) ~ 3/6
P(read  | f. ) ~ 2/6
P(write | f. ) ~ 1/6
```

context γ

Training dataset ***D***

```
f.open(“f2” | “r”);
f.read();
```

```
f.open(“f2” | “w”);
f.write(“c”);
```

```
f.open(“f1” | “r”);
f.read();
```

query:

```
f.open(“file” | “r”);

f. open
```

3-gram model on tokens

Hindle et. al., ACM ICSE’12

```
P(open  | f. ) ~ 3/6
P(read  | f. ) ~ 2/6
P(write | f. ) ~ 1/6
```

context γ

Training dataset ***D***

```
f.open("f2" | "r");
f.read();
```

```
f.open("f2" | "w");
f.write("c");
```

```
f.open("f1" | "r");
f.read();
```

probabilistic model on APIs

Raychev et. al., ACM PLDI'14

```
P(read  | open ) ~ 2/3
P(write | open ) ~ 1/3
```

context γ

query:

```
f.open("file" | "r");

f.[ ? ]
```

3-gram model on tokens

Hindle et. al., ACM ICSE'12

```
P(open  | f. ) ~ 3/6
P(read  | f. ) ~ 2/6
P(write | f. ) ~ 1/6
```

context γ

Training dataset ***D***

```
f.open(“f2” | “r”);
f.read();
```

```
f.open(“f2” | “w”);
f.write(“c”);
```

```
f.open(“f1” | “r”);
f.read();
```

probabilistic model on APIs

Raychev et. al., ACM PLDI’14

```
P(read  | open ) ~ 2/3
P(write | open ) ~ 1/3
```

context γ

query:

```
f.open(“file” | “r”);

f. read
```

3-gram model on tokens

Hindle et. al., ACM ICSE’12

```
P(open  | f. ) ~ 3/6
P(read  | f. ) ~ 2/6
P(write | f. ) ~ 1/6
```

context γ

Training dataset ***D***

```
f.open("f2" | "r");
f.read();
```

```
f.open("f2" | "w");
f.write("c");
```

```
f.open("f1" | "r");
f.read();
```

probabilistic model on APIs

Raychev et. al., ACM PLDI'14

```
P(read  | open ) ~ 2/3
P(write | open ) ~ 1/3
```

context γ

query:

```
f.open("file" | "r");

f.[ ? ]
```

3-gram model on tokens

Hindle et. al., ACM ICSE'12

```
P(open  | f. ) ~ 3/6
P(read  | f. ) ~ 2/6
P(write | f. ) ~ 1/6
```

context γ

What should the context be?

key idea: synthesize a function f:

program context

$\rightarrow \gamma$

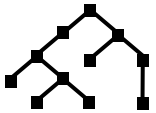
"...All problems in computer science can be solved by **another level of indirection**..."

-- David Wheeler

Creating probabilistic models: our method

["Learning Programs from Noisy Data", ACM POPL'16,
"PHOG: Probabilistic Model for Code", ACM ICML'16,
"Probabilistic Model for Code with Decision Trees", ACM OOPSLA'16]

1. Pick a structure of interest, e.g., ASTs:



---

2. Define a DSL for expressing functions:
(can be Turing complete)

DSL

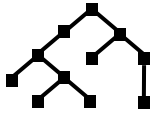
---

3. Synthesize $f_{best} \in$ DSL from Dataset $\boldsymbol{D}$:

$$f_{best} = \text{argmin cost}(\boldsymbol{D}, f)$$
$$f \in \text{DSL}$$

---

4. Use f_{best} to compute context and predict:

f_{best} () $\rightarrow \gamma$

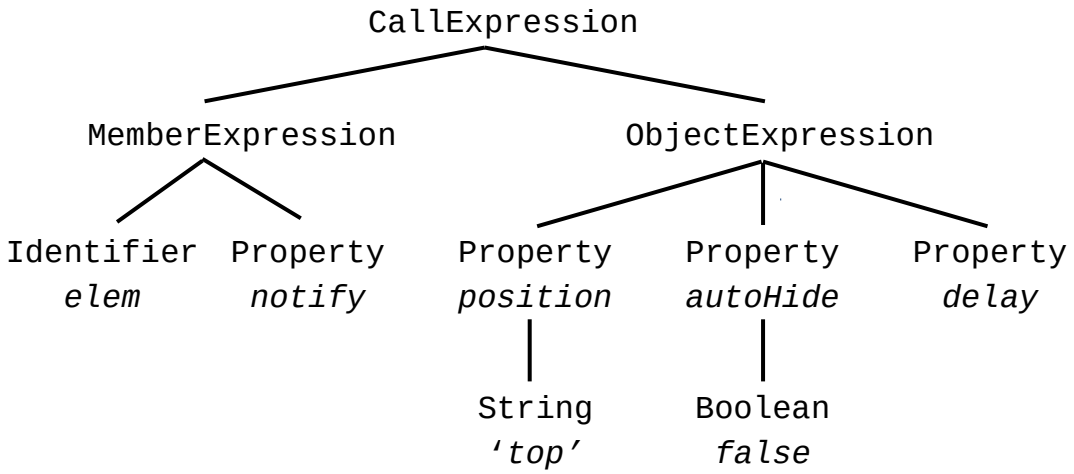
Step 1: Pick Structure of Interest

Let it be abstract syntax trees (ASTs) of programs

JavaScript program

```
elem.notify({
  position: ‘top’,
  autoHide: false,
  delay: 100
});
```

AST



Step 2: Define a DSL over structure

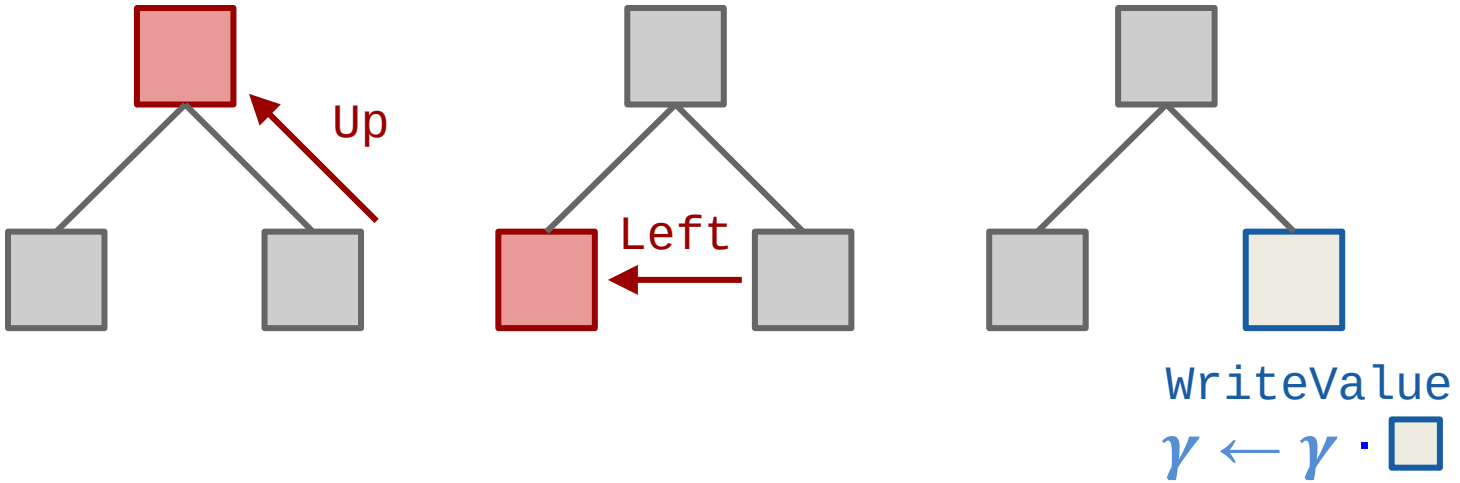
Syntax

```
TCond   ::=  ε | WriteOp TCond | MoveOp TCond

MoveOp  ::=  Up, Left, Right, DownFirst, DownLast,
             NextDFS, PrevDFS, NextLeaf,
             PrevLeaf,PrevNodeType, PrevNodeValue, PrevNodeContext

WriteOp ::=  WriteValue, WriteType, WritePos
```

Semantics



Step 3: synthesize f_{best}

$$f_{best} = \text{argmin cost}(\boldsymbol{D}, f)$$
$$f \in \text{DSL}$$

Step 3: synthesize f_{best}

DSL

```
TCond   ::=  ε | WriteOp TCond | MoveOp TCond
MoveOp  ::=  Up, Left, Right, ...
WriteOp ::=  WriteValue, WriteType, ...
```

generate candidate f

dataset D

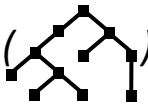
millions ($\approx 10^8$)

Synthesizer

f_{best} = argmin cost(D, f)
$f \in$ DSL

Build Probabilistic Model P

use D and f to compute

$P(element \mid f($  $))$

to scale: iterative synthesis on fraction of examples

$O(|D|)$

$cost(D, f) =$ entropy(P)

Step 4: use f_{best} to predict

program

```
elem.notify(
  ... ,
  ... ,
  {
    position: 'top',
    hide: false,
    ?
  }
);
```

f_{best}	Context γ
Left	{}
WriteValue	{hide}
Up	{hide}
WritePos	{hide, 3}
Up	{hide, 3}
DownFirst	{hide, 3}
DownLast	{hide, 3}
WriteValue	{hide, 3, notify}

{Previous Property, Parameter Position, API name}

Deep3: Experimental Results

[Probabilistic Model of JavaScript]

Dataset **D:** 150,000 files Training Time: ~**100** hours f_{best} ~ **50,000** instr.

Probabilistic Model	**Accuracy** (APIs)
Last two tokens, Hindle et. al. [ICSE'12]	22.2%
Last two APIs, Raychev et. al. [PLDI'14]	30.4%
Deep3	**66.6%**

Details in: "Probabilistic Model for Code with Decision Trees", ACM OOPSLA'16

Deep3: Experimental Results
[Probabilistic Model of Python]

Dataset **D:** 150,000 files Training Time: ~**100** hours f_{best} ~ **120,000** instr

Probabilistic Model	**Accuracy** (identifiers)
Last two tokens, Hindle et. al. [ICSE'12]	38%
Deep3	**51%**

Details in: "Probabilistic Model for Code with Decision Trees", ACM OOPSLA'16

Applying the Concept to Natural Language

[Program Synthesis for Character Level Language Modeling, **ICLR'17 sub**]

Dataset **D:**
Hutter Prize Wikipedia Dataset

Training Time: ~ **8** hours

f_{best} **~ 9,000** instr
uses a char-level DSL with state

Interpretable model, browse here:
http://www.srl.inf.ethz.ch/charmodel.html

Probabilistic Model	**Bits-per-Character**
7-gram (best)	1.94
Stacked LSTM (Graves 2013)	1.67
Char-based DSL synthesis	1.62
MRNN (Sutskever 2011)	1.60
MI-LSTM (Wu et al. 2016)	1.44
HM-LSTM* (Chung et al. 2016)	1.40

Learning (Abstract) Semantics

[Learning a Static Analyzer from Data, **https://arxiv.org/abs/1611.01752**]

```
function isBig(v) {
 return v < this.length
}
[12, 5].filter(isBig);
```

```
VarPtsTo("global", h)
checkIfInsideMethodCall
checkMethodCallName
checkReceiverType
checkNumberOfArguments ...
```

```
VarPtsTo(this, h)
```

Can be understood by experts

Found issues in Facebook's Flow

DEEPCODE

http://deepcode.ai

ETH spin-off, co-founded in 2016 by

Veselin Raychev

Martin Vechev

From code to predictions

Handles any programming language

We are hiring!

More Info

Learning from Large Codebases,
PhD Thesis, ETH Zurich, 2016

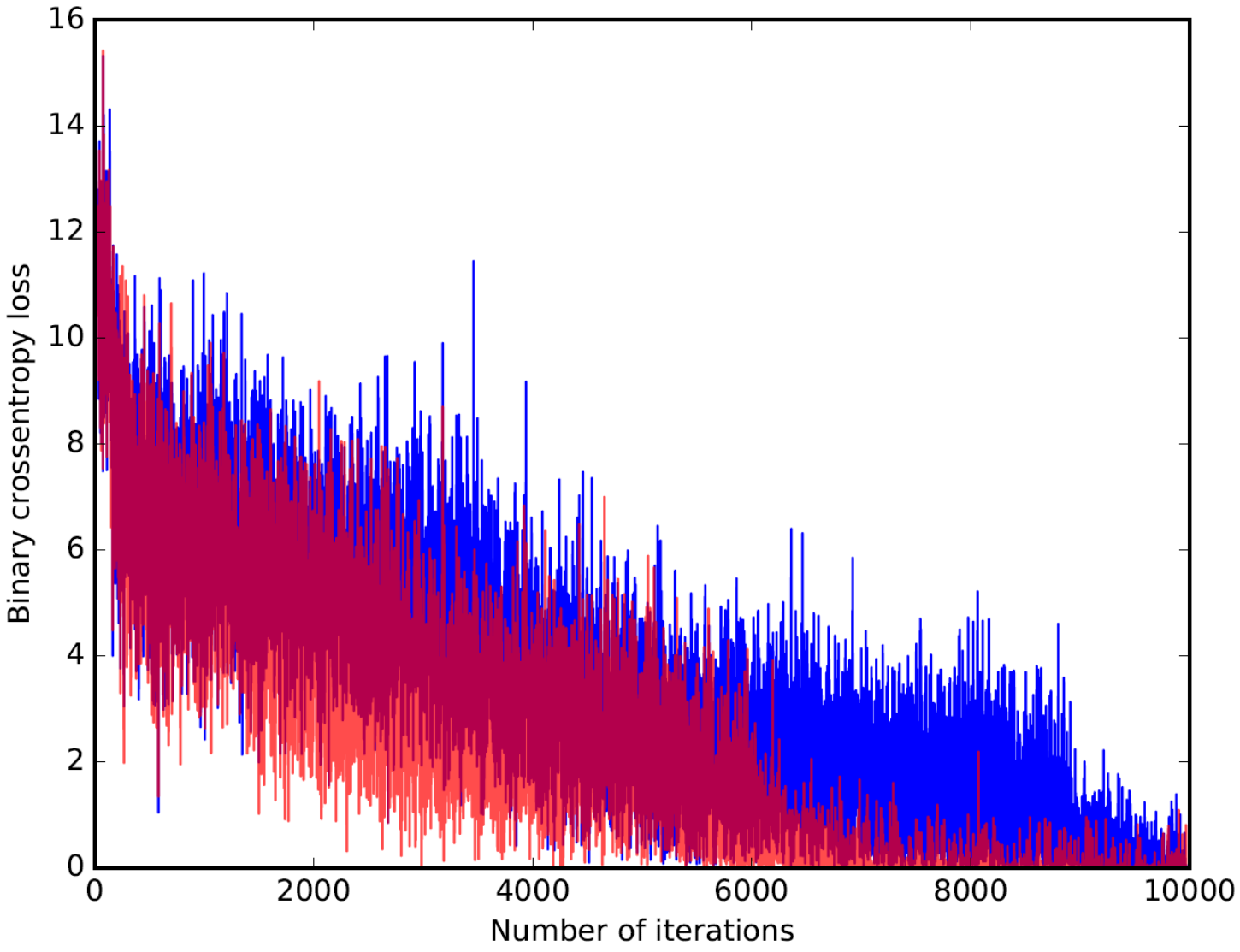
Veselin
Raychev

http://plml.ethz.ch

Dagstuhl Seminar on Big Code Analytics, Nov 2015

Data sets, tools, challenges: http://learningfrombigcode.org

Synthesis with NTMs



Two part talk (22 + 3) ☺

Learning-based Programming Engines

SLANG | JS NICE | NICE 2 Predict | DEGUARD | Deep3

1. Pick a structure of interest, e.g., trees:

2. Define a DSL for expressing functions:
(can be Turing complete)

```
TCond   ::= ε | WriteOp TCond | MoveOp TCond
MoveOp  ::= Up, Left, Right, DownFirst, DownLast,
            NextDFS, PrevDFS, NextLeaf, PrevLeaf,
            PrevNodeType, PrevNodeValue,
WriteOp ::= WriteValue, WriteType, WritePos
```

3. Synthesize $f_{best} \in$ DSL from Dataset D:

$$f_{best} = \underset{f \in \text{DSL}}{\text{argmin}}\ \text{cost}(\boldsymbol{D}, f)$$

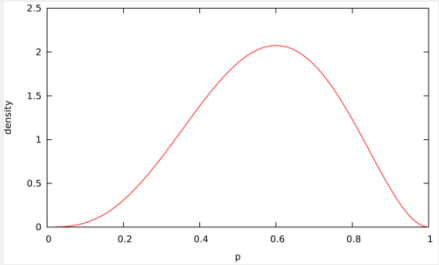
4. Use f_{best} on new structures:

f_{best} () → γ

http://plml.ethz.ch

PSI: Exact Solver for Probabilistic Programs

```
def main() {
 p := Uniform(0,1);
 r := [1,1,0,1,0];
  for  i in [0..r.len] {
  observe(
     Bernoulli(p) == r[i]));
 }
 return p;
}
```

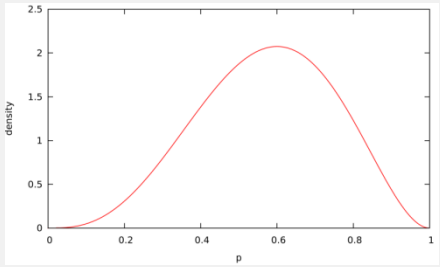


http://psisolver.org

Two part talk (22 + 3) ☺

PSI: Exact Solver for Probabilistic Programs

```
def main() {
 p := Uniform(0,1);
 r := [1,1,0,1,0];
  for  i in [0..r.len] {
  observe(
    Bernoulli(p) == r[i]));
 }
 return p;
}
```

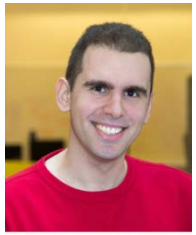


http://psisolver.org

PSI

PSI: Exact Symbolic Inference for Probabilistic Programs, CAV'16

Timon Gehr



Sasa Misailovic

Exact inference for probabilistic programs

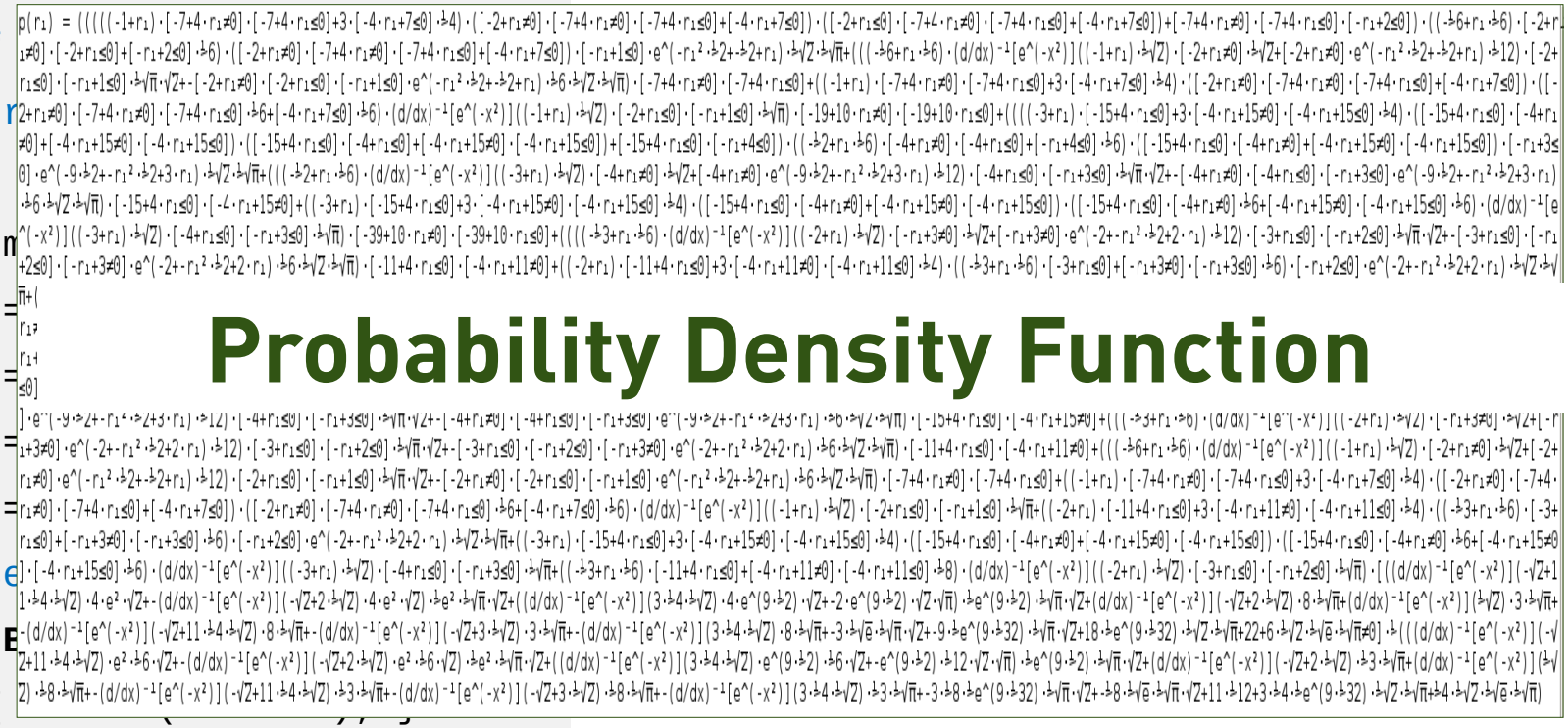
http://psisolver.org/

PSI

```
def max(a,b) {
  r := a;
  if  b > r { r = b; }
    return r;
}
def main() {
 x := Uniform(0,1);
 y := Gauss(0,1);
 z := Uniform(0,1);
 r := max(x,max(y,z));
 observe(x < 0.75);
 if Bernoulli(1/2)
   { assert(r < 0.9); }
 return r + UniformInt(1,3);
}
```

PSI

```
def max(a,b) {
  r := a;
  if
    r
}
def m
 x :=
 y :=
 z :=
 r :=
 obse
 if E
   {
 return r + UniformInt(1,3);
}
```

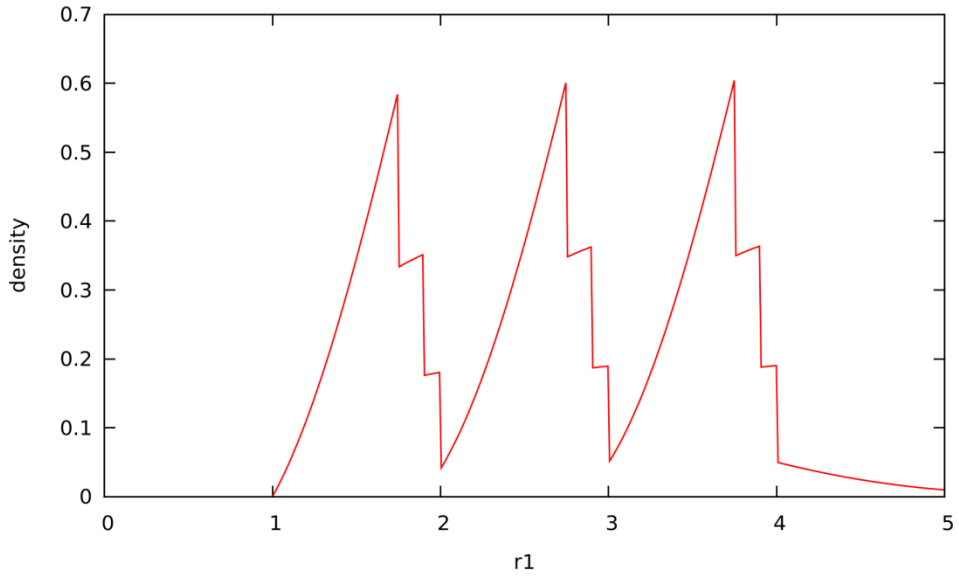


Probability Density Function

PSI

```
def max(a,b) {
  r := a;
  if  b > r { r = b; }
    return r;
}
def main() {
 x := Uniform(0,1);
 y := Gauss(0,1);
 z := Uniform(0,1);
 r := max(x,max(y,z));
 observe(x < 0.75);
 if Bernoulli(1/2)
   { assert(r < 0.9); }
 return r + UniformInt(1,3);
}
```

a little nicer to look at...



can compute various queries on the PDF:
e.g., **expectations**, **marginal probabilities**

E[result] ≈ 2.6929
Pr[error] ≈ 0.132827

PSI: Ingredients

Symbolic Domain for PDFs

$$
\begin{aligned}
e \in E ::= \quad & x \mid \mathsf{e} \mid \pi \mid 0 \mid 1 \mid 2 \mid \dots \\
& \mid \log(e) \mid -e \mid e_1 + \dots + e_n \mid e_1 \cdot \dots \cdot e_n \mid e_1^{e_2} \\
& \mid \delta(e) \mid [e_1 = e_2] \mid [e_1 \le e_2] \mid [e_1 \neq e_2] \mid [e_1 < e_2] \\
& \mid \int_{\mathbb{R}} \mathsf{d}x\, e[\![x]\!] \mid \sum_{x \in \mathbb{Z}} e[\![x]\!] \mid \varphi(e_1, \dots, e_n) \\
& \mid (\mathsf{d}/\mathsf{d}x)^{-1}[\mathsf{e}^{-x^2}](e) \quad \text{(Error function)}
\end{aligned}
$$

Encodes probability density functions

- $\text{Bernoulli}(x; p) = p \cdot \delta(1 - x) + (1 - p) \cdot \delta(x)$
- $\text{Gauss}(x; \mu, \nu) = [\nu = 0] \cdot \delta(x - \mu) + [\nu \neq 0] \cdot \frac{\mathsf{e}^{-(x-\mu)^2/(2\nu)}}{(2\pi\nu)^{\frac{1}{2}}}$
- $\text{UniformInt}(x; a, b) = \frac{\sum_{x' \in \mathbb{Z}} \delta(x - x') \cdot [a \le x'] \cdot [x' \le b]}{\sum_{x' \in \mathbb{Z}} [a \le x'] \cdot [x' \le b]}$

Symbolic Simplification

- Basic algebraic simplifications
  $x + x \to 2 \cdot x, \quad x \cdot x \to x^2, \dots$
- Simplifications on constraints
  $[x = 0] + [x \neq 0] \to 1, \quad [x \le 0] \cdot [0 \le x] \to [x = 0],$
  $\delta(x) \cdot [1 \le x] \to 0, \dots$
- Linearize Guards
  $\delta(y - x^2) \to$
  $[-y \le 0] \cdot ([x = 0] \cdot \delta(y) + [x \neq 0] \cdot \frac{1}{2\sqrt{y}}(\delta(x - \sqrt{y}) + \delta(x + \sqrt{y}))),$
  $\dots$
- Symbolic Integration
  $\int_{\mathbb{R}} \mathsf{d}x \int_{\mathbb{R}} \mathsf{d}y\, [0 \le x] \cdot [x \le 1] \cdot [0 \le y] \cdot [y \le 1] \cdot \delta(z - x \cdot y) \to$
  $-[0 < z] \cdot [z \le 1] \cdot \log(z), \dots$

Learning-based Programming Engines

SLANG | JS NICE | NICE 2 Predict | DEGUARD | Deep3

1. Pick a structure of interest, e.g., trees:

2. Define a DSL for expressing functions:
(can be Turing complete)

```
TCond    ::=  ε | WriteOp TCond | MoveOp TCond
MoveOp   ::=  Up, Left, Right, DownFirst, DownLast,
              NextDFS, PrevDFS, NextLeaf, PrevLeaf,
              PrevNodeType, PrevNodeValue,
WriteOp  ::=  WriteValue, WriteType, WritePos
```

3. Synthesize $f_{best} \in$ DSL from Dataset D:

$$f_{best} = \underset{f \in \text{DSL}}{\text{argmin}}\ \text{cost}(\boldsymbol{D}, f)$$

4. Use f_{best} on new structures:

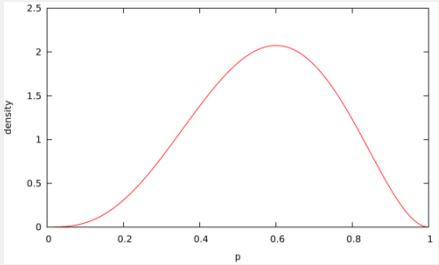
f_{best} () → y

http://plml.ethz.ch

PSI: Probabilistic Solver

PSI SOLVER

```
def main() {
 p := Uniform(0,1);
 r := [1,1,0,1,0];
  for  i in [0..r.len] {
  observe(
    Bernoulli(p) == r[i]));
 }
 return p;
}
```



http://psisolver.org